Introduction to MPI: Lecture 2

Jun Ni, Ph.D. M.E.

Associate Professor
Department of Radiology
Carver College of Medicine

Information Technology Services

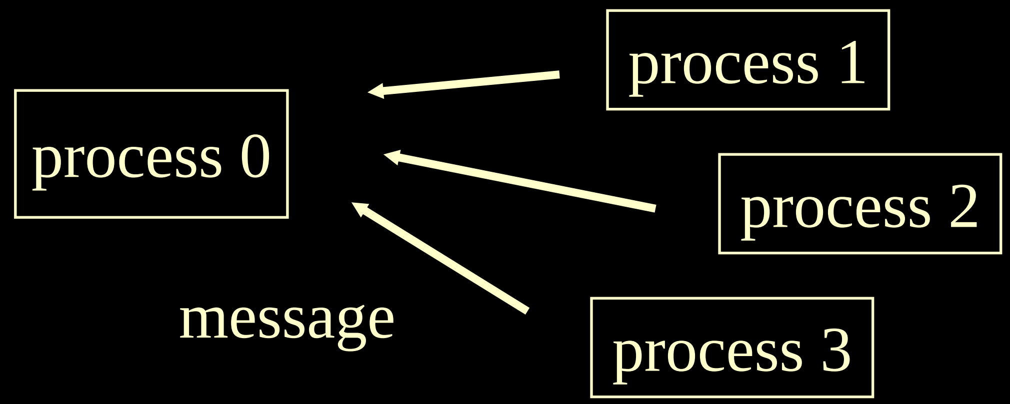
The University of Iowa

Learning MPI by Examples

Simple Greetings Among Processes

Simple Greetings Among Processes

- Example 0: basic communication between processes. Suppose we have p processes
  - p, multiple processes: starting from 0 to p-1
  - process 0 receive messages from other processes



Simple Greetings Among Processes

- Example 0: mechanism
  - system copies the executable code to each processes
  - each process begins execution of the copied executable code
  - different processes can execute different statements by branching within the program based on their ranks (this form of MIMD programming is called single-program multiple-data (SPMD) programming)

Simple Greetings Among Processes

```
/*******************************************************
   greetings.c -- greetings program
    Send a message from all processes with rank != 0 to process 0.
    Process 0 prints the messages received.

   Input: none.
   Output: contents of messages received by process 0.
 *******************************************************/
#include <stdio.h>
#include <string.h>
#include "mpi.h"
```

Simple Greetings Among Processes

```
main(int argc, char* argv[])
{
    int         my_rank;            /* rank of process         */
    int         p;                  /* number of processes  */
    int         source;             /* rank of sender           */
    int         dest;               /* rank of receiver         */
    int         tag = 0;            /* tag for messages          */
    char       message[100];    /* storage for message      */
    MPI_Status  status;         /* return status for receive */

    /* Start up MPI */
    MPI_Init(&argc, &argv);
```

Simple Greetings Among Processes

```
/* Find out process rank  */
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   printf("my_rank is %d\n", my_rank);

   /* Find out number of processes */
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   printf("p, the total number of processes: %d\n",p);

   if (my_rank != 0)
   {
      /* Create message */
   sprintf(message, "Greetings from process %d!", my_rank);
   dest = 0;
```

Simple Greetings Among Processes

```
/* Use strlen+1 so that '\0' gets transmitted */
    MPI_Send(message, strlen(message)+1, MPI_CHAR,
                 dest, tag, MPI_COMM_WORLD);
 }
else        /* my_rank == 0 */
{
   for (source = 1; source < p; source++)
   {   MPI_Recv(message, 100, MPI_CHAR, source, tag,
          MPI_COMM_WORLD, &status);
       printf("%s\n", message);
    }
 }
```

Simple Greetings Among Processes

```
    /* Shut down MPI */
    MPI_Finalize();
}   /* main */
```

Commands:

```
% cc -o greetings greetings.c -lmpi

% /bin/time mpirun -np 8 greetings
```

Simple Greetings Among Processes

Result:

silicon % /bin/time mpirun -np 8 greetings
my_rank is 3
p, the total number of processes: 8
my_rank is 4
p, the total number of processes: 8
my_rank is 0
p, the total number of processes: 8
my_rank is 1
p, the total number of processes: 8
Greetings from process 1!
my_rank is 2

Simple Greetings Among Processes

p, the total number of processes: 8
my_rank is 7
p, the total number of processes: 8
Greetings from process 2!
Greetings from process 3!
my_rank is 5
p, the total number of processes: 8
Greetings from process 4!
Greetings from process 5!
my_rank is 6
p, the total number of processes: 8
Greetings from process 6!
Greetings from process 7!

Simple Greetings Among Processes

real 1.501
user 0.005
sys 0.049

Simple Greetings Among Processes

- Example 0: (in Fortran)

```
c  greetings.f -- greetings program
c
c  Send a message from all processes with rank != 0 to process 0.
c     Process 0 prints the messages received.
c
c  Input: none.
c  Output: contents of messages received by process 0.
c
c  Note:  Due to the differences in character data in Fortran and char
c      in C, their may be problems in MPI_Send/MPI_Recv
c
```

```
      program greetings
c
      include 'mpif.h'
c
      integer my_rank
      integer p
      integer source
      integer dest
      integer tag
      character*100 message
      character*10 digit_string
      integer size
      integer status(MPI_STATUS_SIZE)
      integer ierr
c
```

```
c   function
    integer string_len
c
    call MPI_Init(ierr)
c
    call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
    call MPI_Comm_size(MPI_COMM_WORLD, p, ierr)
c
    if (my_rank.ne.0) then
       call to_string(my_rank, digit_string, size)
       message = 'Greetings from process ! '  // digit_string(1:size) +//
       dest = 0
       tag = 0
      call MPI_Send(message, string_len(message),
        MPI_CHARACTER, dest, tag, MPI_COMM_WORLD, ierr)
   else
```

```
      do 200 source = 1, p-1
      tag = 0
      call MPI_Recv(message, 100, MPI_CHARACTER, source,
     +     tag, MPI_COMM_WORLD, status, ierr)
      call MPI_Get_count(status, MPI_CHARACTER, size, ierr)
      write(6,100) message(1:size)
 100  format(' ',a)
 200     continue
       endif
c
        call MPI_Finalize(ierr)
        stop
        end
c
c
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c   Converts the integer stored in number into an ascii
c   string.  The string is returned in string.  The number of
c   digits is returned in size.
        subroutine to_string(number, string, size)
        integer number
        character *(*) string
        integer size

        character*100 temp
        integer local
        integer last_digit
        integer i

        local = number
       i = 0
```

```
c   strip digits off starting with least significant
c   do-while loop
 100         last_digit = mod(local,10)
             local = local/10
             i = i + 1
             temp(i:i) = char(last_digit + ichar('0'))
          if (local.ne.0) go to 100

          size = i

c   reverse digits
          do 200 i = 1, size
             string(size-i+1:size-i+1) = temp(i:i)
 200      continue
c
          return
          end
```

```
c   to_string
c
c
ccccccccccccccccccccccccccccccccccccccccccccccccc
c   Finds the number of characters stored in a string
c
          integer function string_len(string)
          character*(*) string
c
          character*1 space
          parameter (space = ' ')
          integer i
c
          i = len(string)
```

```
c   while loop
 100   if ((string(i:i).eq.space).and.(i.gt.1)) then
              i = i - 1
          go to 100
          endif
c
          if ((i.eq.1).and.(string(i:i).eq.space)) then
              string_len = 0
          else
              string_len = i
          endif
c
          return
          end
c   end of string_len
```

```
f77 -o greetings greetings.f -lmpi
/bin/time mpirun -np 8 greetings
```

```
Greetings from process 1!
Greetings from process 2!
Greetings from process 3!
Greetings from process 4!
Greetings from process 5!
Greetings from process 6!
Greetings from process 7!

real 1.717
user 0.005
sys  0.040
```

Simple Greetings Among Processes

- Anotomy of the first example
  - user issues a directive to the operating system that has effect of placing a copy of the executable program on each processor
  - each processor begins execution of its copy of the executable code
  - different processes can execute different statements by branching within the program base don their process ranks

Simple Greetings Among Processes

- MPI is not a programming language
- MPI is just a parallel library which contains many definitions of functions or subroutines
- MPI has its own data types with MPI_ identifier and data-type definition in upper cases, such as

MPI Data-types

- MPI_CHAR,
- MPI_SHORT, MPI_INT, MPI_LONG,
- MPI_UNSIGNED_CHAR, MPI_UNSIGNED
- MPI_UNSIGNED_SHORT, MPI_UNSIGNED_LONG,
- MPI_FLOAT,
- MPI_DOUBLE, MPI_LONG_DOUBLE
- MPI_BYTE,
- MPI_PACKED,
- MPI_LONG_LONG_INT

Simple Greetings Among Processes

- MPI_Init() must be called before other MPI functions are invoked.
- MPI_Finalize() must be called after the program is finished.

Simple Greetings Among Processes

- MPI_Comm_rank() function returns the rank of a process in its second parameter.
- Syntax:

```
MPI_Comm_rank (    MPI_Comm comm      /*in */,
                   int* size          /* out */)
```

-comm --- inter-communicator, ground or collection of process
The function returns the rank in the group.
Default value of comm is MPI_COMM_WORLD, all processes during execution

Simple Greetings Among Processes

- MPI_Comm_size() function returns the number of processes in its second parameter.
- Syntax:

```
MPI_Comm_size (     MPI_Comm comm      /*in */,
                    int* size          /* out */)
```

-comm --- inter-communicator, group or collection of process
The function returns the total number of processes in the group.
Default value of comm is MPI_COMM_WORLD, all processes during execution

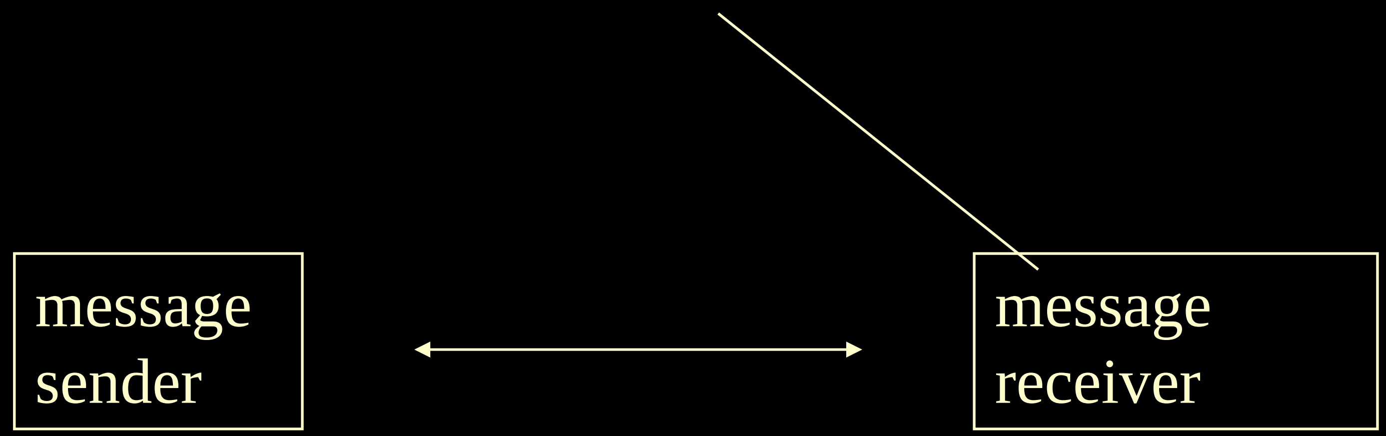
Simple Greetings Among Processes

- MPI_Send and MPI_Recv() functions are the most basic message-passing commands in MPI library
- Review basic message passing mechanism

compose message (letter); put in an envelop;
stop by a poster office for stamping; drop to the mail box;
Add more information about receiver's address, size, and subject

Simple Greetings Among Processes

receive message (letter); distinguish the priority; sorting message; reply address; action, and return message back;



Key points: message subject, message format, message size

Simple Greetings Among Processes

- Solutions to message passing
  - each process sends two messages: one for method and another for actual message content
  - each processes send single message which contains both information. It should be encoded before sending and decoded after receiving.
  - tag communication signal with the envelop being sent out. MPI has its own tag identification numbers

Simple Greetings Among Processes

- Communicator can specify the scope of process activities
  - Two processes using distinct communicator can not receive messaged from each other.
- The complete message passing envelop contains
  - the rank of the receiver
  - the rank of the sender
  - a tag
  - communicator

Simple Greetings Among Processes

– MPI_Send() syntax:

```
int MPI_Send (    void* message        /*in */,
                  int count            /* in */,
                  MPI_Datatypes        /*in */,
                  int dest             /*in*/,
                  int tag              /*in*/,
                  MPI_Comm comm        /*in*/)
```

Simple Greetings Among Processes

– MPI_Recv() syntax:

```
int MPI_Recv (     void* message          /*out */,
                   int count              /* in */,
                   MPI_Datatypes          /*in*/,
                   int source             /*in*/,
                   int tag                /*in*/,
                   MPI_Comm comm          /*in*/,
                   MPI_Status* status     /*out*/)
```

Simple Greetings Among Processes

- The content of the massage are stored in a block of memory referenced by the variable message (In C it is a char array, while in Fortran it is a char variable.)
- Count and MPI_Datatype specify how much allocated storage is needed for the message.
  - The amount of space allocated for receiving buffer does not have to match the exact amount of space the message being received
  - Make sure that there is sufficient storage allocated for receiving

Simple Greetings Among Processes

- The integer parameters "dest" in MPI_Send() and "source" in MPI_Recv() are, respectively, the ranks of the receiving and the sending processes.
  - dest in MPI_Send() indicates the receiving process
  - source in MPI_Recv() indicates the sending process
  - MPI_ANY_SOURCE can be used for any sending process rather than a particular sending process

Simple Greetings Among Processes

- Parameter tag and comm are, respectively, the tag and communicator.
  - tag is a integer variable, specification of message passing mode
  - comm is the communicator, specification of collection of message passing process
  - In this example, tag is 0 and comm is MPI_COMM_WORLD, indicating all running processes during execution
  - MPI_ANY_TAG can be used in MPI_Recv() for any tag.

Simple Greetings Among Processes

- For example process A sends a message to process B
  - comm, which the process A uses, in its call to MPI_Send() must ne identical to the argument that B uses in its call to MPI_Recv(), while A must use a tag and B can receive with either an identical tag or MPI_ANY_TAG

identical or MPI_ANY_TAG

MPI_Send(, , , ,tag, comm)

MPI_Recv(, , , , tag, coom)

Sending process → Receiving process

Simple Greetings Among Processes

- status of MPI_Status in MPI_Recv() returns information on the data that was actually received.
  - status is a variable of structure, defined as MPI_Status, which has three members, one for source, one for tag and one for error code
    - status->MPI_SOURCE
    - status->MPI_TAG
    - status->MPI_ERROR

Simple Greetings Among Processes

- Either MPI_Send() or MPI_Recv() returns a error code in C, while the error code, passed back from the called subroutine to the calling code, is located as the last argument of the subroutine in Fortran with MPI.

Simple Greetings Among Processes

- Exercise:

1. Modify greetings.c (or greetings.f) so that process 0 send a "string" message to all the other processes. The receiving processes receive and then print the message on screen.

2. Modify greetings.c (or greetings.f) so that process u send a "integer" message to processes v and w. The v and w calculate the square and cubic values, respectively, based on the received integer. They print out the values.